Project Implementation
COS 301 Buzz Project
Group: Testing Phase: Resources
Version 1.0

Carla de Beer 95151835
Prenolan Govender 13102380
Shaun Meintjes 13310896
Collins Mphahlele 12211070
Dumisani Msiza 12225887
Joseph Murray 12030733
Sifiso Shabangu 12081622
Joseph Potgieter 12003672
Johan van Rooyen 11205131

https://github.com/Carla-de-Beer/Testing-Resources

University of Pretoria

24 April 2015

Contents

1 Introduction

The purpose of this task was to test the functionality provided by the Resources teams for the Buzz Project.

According to the master specification document, version 0.1, released 13 March 2015 (SOLMS *et al*, 2015), the buzzResources module is used to upload and manage resources like media (e.g. images, video) and documents (e.g. PDF documents, Open Document Format documents). These resources are to be either embedded or linked to in posts.

This team took each of the pre and post-conditions of the required use case, considering the work of both Resources A and Resources B, and tested them for compliance with the functional requirements.

The use case for this team is called **uploadResource**. In terms of the requirements for this use case, and as defined by the master specification document, users should be able to upload resources such as media files or documents. Any uploaded resource should be accessible by other users who should also be able to specify links to that resource.

The functional requirements preconditions for this use case included the need to

- detect the mime type
- check that size constraints are met
- check that the resource type is supported

The functional requirements postconditions for this use case included the need to

- check that the resource persisted
- check that the URL for the resource was created

2 Functional Testing

2.1 Resources A

2.1.1 Pre-condition violations

getResourcesBySpaceId The pre-condition will be that the SpaceID that is sent in as a parameter does exist in the database prior to this execution. There are no violations of this pre-condition.

getResourcesAll The pre-condition will be that there exist resources in the database prior to this execution. There are no violations of this pre-condition.

getResourcesRelated The pre-condition will be that the appraisal type (RelatedID) that is sent in as a parameter does exist in the database prior to this execution. There are no violations of this pre-condition.

removeResource The pre-conditions which in this case is resource ID being passed as a parameter to removeResource, the ID must exist in the databaase for a resource to be removed.

addResourceType The pre-condition will be the MIME type passed would be a valid and existing MIME type and the size limit would be a valid size which would be stated in bytes.

The pre-condition can be violated by passing the MIME type that does not exist and the function will just add the size limit and the MIME type to the Resources contraints in the database.

removeResourceType The objectID passed as a parameter must conform to a hexadecimal sequence. If the objectID specified is not hexadecimal, the function returns false.

updateResourceType The pre-conditions which in this case is the constraintID passed as a parameter which is the ID of the constraint to be updated must exist in Resources contraints in the database and sizeLimit which is the new size limit of the constraint must be stated in bytes.

2.1.2 Post-condition violations

getResourcesBySpaceId The post-condition will be that the resources that exist in a certain space will be returned. This post-condition fails, because the function queries the database and returns all of the resources in the database, which is incorrect. No exceptions are thrown if the Buzz Space does not exist.

getResourcesAll The post-condition will be that all of the resources in the database will be returned. This post-condition holds, and all of the resources are returned.

getResourcesRelated The post-condition will be that the resources that exist in the database of the specified appraisal type will be returned. This post-condition fails, because the function queries the database and returns all of the resources in the database, which is incorrect.

removeResource The post condition will be the removal of the specified resource by ID in the database.

The removeResource does not conform to the post-condition as it cannnot remove the resource that exist in the database with the specified ID.

addResourceType (addConstraint) The post-condition is when the size limit and the MIME type would be added to the Resources contraints in the database.

removeResourceType (removeConstraint) The objectID which is the ID of the constraint to be removed from Resources contraints in the database will be removed.

updateResourceType (updateConstraint) The constraint with the specified constraintID in Resources contraints in the database will be updated by the new size limit.

2.1.3 Data structure requirements

getResourcesBySpaceId No data structure requirements mentioned in the service contract, nor does getResourceBySpaceId utilise any data structures.

getResourcesAll No data structure requirements mentioned in the service contract, nor does getResourceAll utilise any data structures..

getResourcesRelated No data structure requirements mentioned in the service contract, nor does getResourcesRelated utilise any data structures.

2.2 Resources B

None of the functions or use cases for Resources B make use of callbacks.

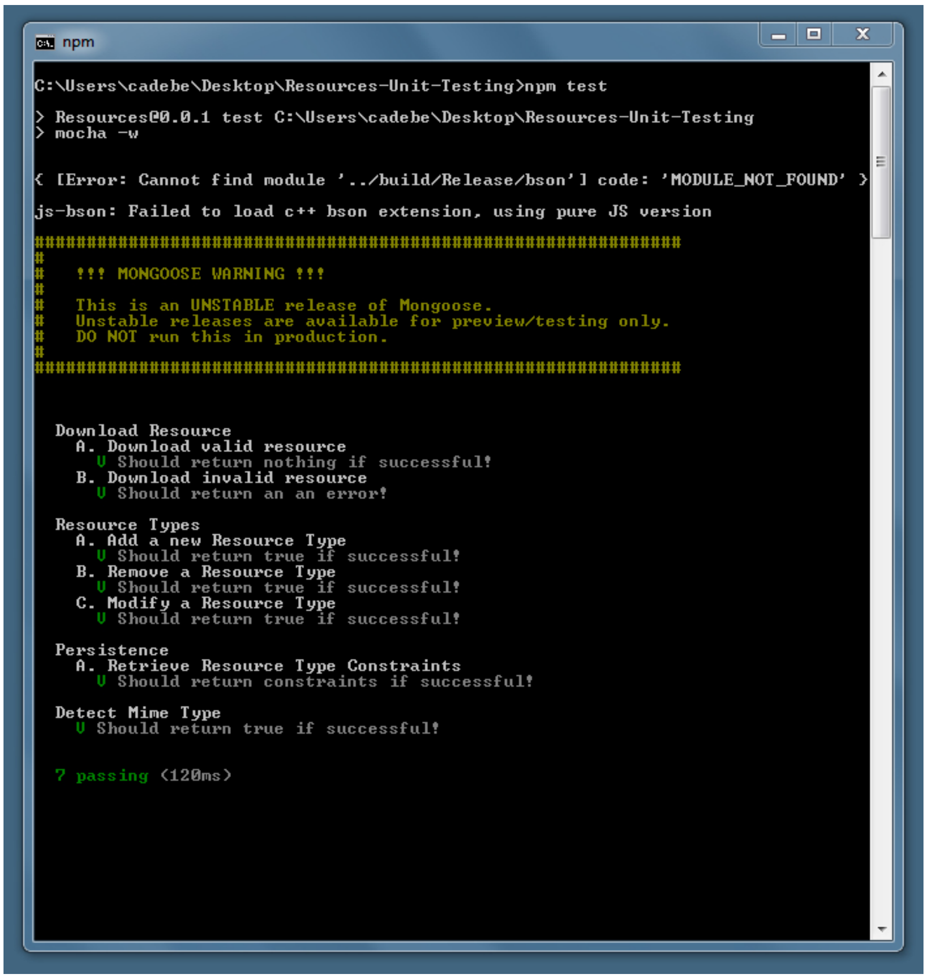
"Nearly everything in node uses callbacks [...] Callbacks are functions that are executed asynchronously, or at a later time. Instead of the code reading top to bottom procedurally, async programs may execute different functions at different times based on the order and speed that earlier functions like http requests or file system reads happen."

The implementation code makes an attempt to allow for the creation of modules for export, but since all the functions, including the main use case, are all object or value-returning functions, it can be said that the code fails, outright, all its services contracts. This is due to the fact that the Resources B code, as it currently stands, cannot be unit tested or integrated into the main code framework for the project. The Testing Resources B team undertook an attempt to write unit testing code, working with the existing code structure, rather than amending it other than by introducing a callback to the `uploadResources` function, in order to be able to test the code for compliance with both functional and non-functional requirements. The code did not run and terminated in an error message stating `"Error: Cannot find module 'DataBaseStuff'"`. It seems, therefore that one of the modules is missing from the code structure and, as such, the code could not be tested by means of callbacks. After some rummaging around in the various GitHub branches, squirrelled away in one of the team members' branches was a section called "Unit Testing Code". It became clear that this code is intended to be run via node package manager (npm) and ran unit tests for the following options:

- Download valid resource
- Download invalid resource

- Add a new Resource Type
- Remove a Resource Type
- Modify a Resource Type
- Retrieve Resource Type Constraints
- Detect Mime Type

In the absence of being able to test via modules and callbacks, the testing team used this unit test code, as provided by the implementation team, for the testing the Resources B section. Below is a screenshot of the result obtained from the initial running of the provided unit testing code, prior to changing input parameters.



```
C:\Users\cadebe\Desktop\Resources-Unit-Testing>npm test

> Resources@0.0.1 test C:\Users\cadebe\Desktop\Resources-Unit-Testing
> mocha -w

{ [Error: Cannot find module '../build/Release/bson'] code: 'MODULE_NOT_FOUND' }

js-bson: Failed to load c++ bson extension, using pure JS version
##############################################################
#
#   !!! MONGOOSE WARNING !!!
#
#   This is an UNSTABLE release of Mongoose.
#   Unstable releases are available for preview/testing only.
#   DO NOT run this in production.
#
##############################################################


  Download Resource
    A. Download valid resource
      √ Should return nothing if successful!
    B. Download invalid resource
      √ Should return an an error!

  Resource Types
    A. Add a new Resource Type
      √ Should return true if successful!
    B. Remove a Resource Type
      √ Should return true if successful!
    C. Modify a Resource Type
      √ Should return true if successful!

  Persistence
    A. Retrieve Resource Type Constraints
      √ Should return constraints if successful!

  Detect Mime Type
    √ Should return true if successful!


  7 passing (120ms)
```

Figure 1: Unit testing Resources B: the output provided by running the unit test code, as provided by the implementation team

2.2.1 Pre-condition violations

uploadResources The uploadResources use case function makes an attempt to fulfil the mimetype detection precondition, as well as the `getResourceTypeConstraints` and throws an exception in the case where

- the file size constraints have not been met
- the resource type is not supported
- the mimetype could not be detected

The raising of these exceptions is commensurate with the services contract requirements, as specified in Figure 28 in the master specification document (SOLMS et al, 2015). There is no actual unit testing code written for the `uploadResources` use case. Considering that the other functions pertaining to the uploading of resources do not work, it is expected that this function will fail. In terms of the actual code, the mimetype size is tested by means of the line:

```
file.size <= (constraints.maximumSize * 1000)
```

There is no explanation as to why this particular value has been chosen as the file size constraints. Something similar to the notion of a constant value, declared at the head to the file, might have been more appropriate and explanatory in this case. In summary, there is an attempt to fulfil the use case pre-conditions by means of inclusion of exceptions for the cases where the preconditions may fail, however, the code fails in cases where erroneous input is added. In such cases the code would have benefited from the application of additional exception throwing to cover these scenarios.

downloadResource This use case calls the line

```
var results = resources.downloadResource("mountain.jpg");
```

and when this line is changed to read

```
var results = resources.downloadResource("mountain");
```

the code still executes successfully. It appears that the unit testing for this function is not working correctly, and as such, the use case cannot be tested with the code provided.

removeResource The pre-condition of removeResource is assumed to refer to the identification number of a resource in the database which should exist prior to this execution. There are no violations of this pre-condition due to error checks being performed to verify whether or not the entry exists.

detectMimeType There appears to be problems with the mimetype detection functionality. It is hard to see how this function, as it currently stands, will return a value other than true. Running the provided test code for the "Detect Mime Type" seems to confirm this. Each time the code was run, the code executes without fail when tested against various mimetypes. Representatives from the various mimtype groups were used, with emphasis on the ones that are expected to be most frequently used, including:

- image/jpeg

- image/png
- image/gif
- image/targa
- image/svg+xml
- image/example
- audio/mp3
- message/http
- multipart/alternative
- multipart/encrypted
- text/cmd
- text/csv

However, when the line `var results = mime.detectMimeType("./r/mountain.jpeg");` was amended to `var results = mime.detectMimeType("./r/rubbish");` the code also happily returned a true value. It seems that neither the the unit testing code for this use case, not the actual use case code, works correctly in that the mimetypes are not correctly detected.

2.2.2 Post-condition violations

removeResource The post-condition of removeResource would be the removal of an entry. Due to a lack of unit testing functionality, the post-condition fails as there is no visible indication of removeResource working.

2.2.3 Data structure requirements

removeResource No data structure requirements mentioned in the service contract, nor does removeResource utilise any data structures.

3 Non-functional testing / assessment

3.1 Resources A

The use case was tested against the following list of architectural requirements:

3.1.1 Usability

getResourcesBySpaceId This function is easy and simple to use. The function is called with the Space ID, and a list of the resources for that space is returned.

getResourcesAll This function is simple and easy to use. The function is called and all resources within the database is returned.

getResourcesRelated This function is simple and easy to use. The function is called and an appraisal type is sent as parameter. All resources of that appraisal type is returned.

3.1.2 Performance

getResourcesBySpaceId This function does not suffer from performance issues. The performance of this function depends on the connection and communication speed of the database.

getResourcesAll This function does not suffer from performance issues. The performance of this function depends on the connection and communication speed of the database.

addingConstraint,removeConstraint and updateConstraint Connection to the database takes longer adding Constraint response time is unrealistic for the system as it takes longer.

getResourcesRelated This function does not suffer from performance issues. The performance of this function depends on the connection and communication speed of the database.

3.1.3 Scalability

getResourcesBySpaceId This function does not place any restriction on the possible size of the database, as a list of all resources within a certain space is returned.

getResourcesAll This function does not place any restriction on the possible size of the database, as a list of all resources in the database is returned.

getResourcesRelated This function does not place any restriction on the possible size of the database, as a list of resources of a certain type is returned.

3.1.4 Testability

getResourcesBySpaceId This function is easy to test, as it can be tested from the interfaces that was created.

getResourcesAll This function is easy to test, as it can be tested from the interfaces that was created.

getResourcesRelated This function is easy to test, as it can be tested from the interfaces that was created.

addingConstraint,removeConstraint and updateConstraint No unit test created for testing of each functions implemented and provided testing is in the app.js file which caters the interface side of the function,however adding Constraint and other functions result in too long to response exception being thrown by the browser as connection with the database is not well established and fails.

3.1.5 Security

getResourcesBySpaceId This function is secure, as it only retrieves information from the database, about the resources for a certain space.

getResourcesAll This function is secure, as it only retrieves information from the database, about all of the resources in the database.

getResourcesRelated This function is secure, as it only retrieves information from the database, about all of the resources of a certain appraisal type.

addingConstraint Interface is not secured in the client side adding Constraint section does not validate user input and adds the given input to the database without any validation ,the only validation is on the server side where it checks if the input is not already in the database.

3.1.6 Reliability

getResourcesBySpaceId This function is very unreliable, because it does not conform to the post-condition. It returns a list of all of the resources, instead of only returning a list of resources within a certain space.

getResourcesAll This function is reliable as it conforms to the post-condition. The function returns a list of all of the resources within the database.

getResourcesRelated This function is very unreliable, because it does not conform to the post-condition. It returns a list of all of the resources, instead of only returning a list of resources of a certain appraisal type.

addingConstraint,removeConstraint and updateConstraint Lack of input validation results in garbage data entered in the database,which result in unreliable results and connection to database takes too long or results in browser throwing an exception for taking too long to respond,thus module is not reliable to complete a task base on this factors.

3.1.7 Reusability

getResourcesBySpaceId This function is reusable, as it retrieves information from the database, about the resources for a certain space. This function can be called and reused throughout the system.

getResourcesAll This function is reusable, as it only retrieves information from the database, about all of the resources in the database. This function can be called and reused throughout the system.

getResourcesRelated This function is reusable, as it only retrieves information from the database, about all of the resources of a certain appraisal type. This function can be called and reused throughout the system.

3.1.8 Pluggability

getResourcesBySpaceId This function is pluggable, as it retrieves information from the database, about the resources for a certain space. This function can be called and reused within any system if called on a database that has the same structure as the current database.

getResourcesAll This function is reusable, as it only retrieves information from the database, about all of the resources in the database. This function can be called and reused within any system if called on a database that has the same structure as the current database.

getResourcesRelated This function is reusable, as it only retrieves information from the database, about all of the resources of a certain appraisal type. This function can be called and reused within any system if called on a database that has the same structure as the current database.

3.2 Resources B

The use case was tested against the following list of architectural requirements:

3.2.1 Usability

removeResource removeResource demonstrates usability by being simple and easy to use. The function only needs be called along with the ID of the resource to be removed which is intuitive.

3.2.2 Performance

removeResource removeResource does not suffer from any performance issues in and of itself, however should the database maintain a sizeable stature, removeResource will be seen to suffer in performance. This is to be expected. performance

3.2.3 Scalability

removeResource removeResource offers no solutions to accomodate an increase or decrease in size of the database.

3.2.4 Testability

removeResource removeResource has a complete lack of testability due to it not conforming to proper unit testing standards (making use of callbacks, depending on a live database etc.).

3.2.5 Security

removeResource The only security concern would be an incorrect ID being used in order to remove a different resource than required, potentially putting the system at risk.

3.2.6 Reliability

removeResource According to the failure of the post-condition, removeResource gives a poor indication with regards to reliability.

3.2.7 Reusability

removeResource removeResource is not very reusable due to the dependence on a single database. Using this function throughout this system would not be possible/practical.

3.2.8 Pluggability

removeResource In the same vein as reusability, removeResource cannot be integrated into other systems due to a lack of robustness and versatility.

4 Critical Evaluation and Recommendations

4.1 Resources A

4.1.1 Master specification compilance

The following of the master specification for the implementation was followed correctly to some extent in the resource implementation ,most functions were implemented and worked to some extent,functionality was prioritised over following software principles like having a unit test for each function created and being specific in documentation of functions parameter types .

4.2 Resources B

4.2.1 Code and code structure legibility

The code has some degree of JSDoc comments, but could have benefited from a more comprehensive commenting system so as to allow newcomers to the code to better understand the program logic. The use of an explanatory README file would also have been welcomed by the testing team and others new to the work to help understand the logic behind the system implemented.

Thus, from a code legibility point of view, there are some improvements that can be made to make the existing code more user-friendly. On a positive note, the code does make use of modularisation in the sense the additional functionalities are separated out from the main use case, making it easier to understand, and perhaps also easier to test the code. In terms of the files themselves, the file structure could have benefited from a better organisational structure. There are files with duplicate naming, for example, residing in different folders, making it unclear how these fit into the existing code structure (it was assume that the code folders in the master branch were the most current). In addition, the unit testing code was hidden away in a branch and, as such, was not that straightforward to find.

5 References

SOLMS, F., PIETERSE V., OMELEZE S., RAMASILA, L. 2015. *Buzz Discussion Board Requirements and Design Specifications (version 0.1)*. Department of Computer Science, University of Pretoria.